LAB#13

POINTERS IN C++ PROGRAMMING

Pointers are powerful features of C++. They are used in C++ program to access the memory and manipulate the address.

Address in C++

To understand pointers, we should first know how data is stored on the computer.

Each variable you create in your program is assigned a location in the computer's memory. The value the variable store is actually stored in the location assigned. To know where the data is stored, C++ has an **&** operator. The **& (reference)** operator gives you the address occupied by a variable.

If var is a variable then, **&var** gives the address of that variable.

Consider the following which will print the address of the variables defined −

```
#include <iostream>

using namespace std;
int main ()
{
  int var1;
  char var2[10];

  cout << "Address of var1 variable: ";
  cout << &var1 << endl;

  cout << "Address of var2 variable: ";
  cout << &var2 << endl;

  return 0;
}
```

When the above code is compiled and executed, it produces the following result −

Address of var1 variable: 0xbfebd5c0
Address of var2 variable: 0xbfebd5b6

What are Pointers?

A pointer is a variable whose value is the address of another variable. Like any variable or constant, you must declare a pointer before you can work with it. The general form of a pointer variable declaration is −

*type *var-name;*

Here, type is the pointer's base type; it must be a valid C++ type and var-name is the name of the pointer variable. The asterisk you used to declare a pointer is the same asterisk that you use for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Following are the valid pointer declaration −

```
int   *ip;   // pointer to an integer
double *dp; // pointer to a double
float *fp;   // pointer to a float
char  *ch     // pointer to character
```

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

Using Pointers in C++

There are few important operations, which we will do with the pointers very frequently.
(a) We define a pointer variable.
(b) Assign the address of a variable to a pointer.
(c) Finally access the value at the address available in the pointer variable. This is done by using unary operator * that returns the value of the variable located at the address specified by its operand. Following example makes use of these operations −

```
#include <iostream>
using namespace std;int main ()
{
int  var = 20;  // actual variable declaration.

int  *ip;       // pointer variable

ip = &var;       // store address of var in pointer variable

cout << "Value of var variable: ";

cout << var << endl;

 // print the address stored in ip pointer variable
```

```
  cout << "Address stored in ip variable: ";
  cout << ip << endl;

  // access the value at the address available in pointer
  cout << "Value of *ip variable: ";
  cout << *ip << endl;

  return 0;
}
```

When the above code is compiled and executed, it produces result something as follows −

```
Value of var variable: 20
Address stored in ip variable: 0xbfc601ac
Value of *ip variable: 20
```

Pointers in C++

Pointers have many but easy concepts and they are very important to C++ programming. There are following few important pointer concepts which should be clear to a C++ programmer –

1. Null Pointers

It is always a good practice to assign the pointer NULL to a pointer variable in case you do not have exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a null pointer. The NULL pointer is a constant with a value of zero defined in several standard libraries, including iostream. Consider the following program

```
#include <iostream>

using namespace std;
int main () {
  int  *ptr = NULL;
  cout << "The value of ptr is " << ptr ;

  return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
The value of ptr is 0
```

2. Pointer Arithmetic

As we understood pointer is an address which is a numeric value; therefore, we can perform arithmetic operations on a pointer just as you can a numeric value. There are four arithmetic operators that can be used on pointers: ++, --, +, and -.
To understand pointer arithmetic, let us consider that ptr is an integer pointer which points to the address 1000. Assuming 32-bit integers, let us perform the following arithmetic operation on the pointer −

ptr++

It will point to the next integer. This operation will move the pointer to next memory location without impacting actual value at the memory location. If ptr points to a character whose address is 1000, then above operation will point to the location 1001 because next character will be available at 1001.

Incrementing a Pointer

We prefer using a pointer in our program instead of an array because the variable pointer can be incremented, unlike the array name which cannot be incremented because it is a constant pointer. The following program increments the variable pointer to access each succeeding element of the array −

```
#include <iostream>

using namespace std;
const int MAX = 3;

int main () {
  int  var[MAX] = {10, 100, 200};
  int  *ptr;
  // let us have array address in pointer.
  ptr = var;

  for (int i = 0; i < MAX; i++) {
    cout << "Address of var[" << i << "] = ";
    cout << ptr << endl;

    cout << "Value of var[" << i << "] = ";
    cout << *ptr << endl;

    // point to the next location
    ptr++;
  } return 0;
}
```

When the above code is compiled and executed, it produces result something as follows −

```
Address of var[0] = 0xbfa088b0
Value of var[0] = 10
Address of var[1] = 0xbfa088b4
Value of var[1] = 100
Address of var[2] = 0xbfa088b8
Value of var[2] = 200
```

3. Pointers vs Arrays

Pointers and arrays are strongly related. In fact, pointers and arrays are interchangeable in many cases. For example, a pointer that points to the beginning of an array can access that array by using either pointer arithmetic or array-style indexing. Consider the following program −

```
#include <iostream>
using namespace std;
const int MAX = 3;

int main () {
  int  var[MAX] = {10, 100, 200};
  int  *ptr;
  // let us have array address in pointer.
  ptr = var;

  for (int i = 0; i < MAX; i++) {
    cout << "Address of var[" << i << "] = ";
    cout << ptr << endl;

    cout << "Value of var[" << i << "] = ";
    cout << *ptr << endl;

    // point to the next location
    ptr++;
  }
  return 0;
}
```

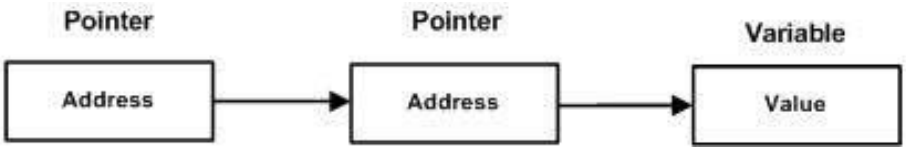
When the above code is compiled and executed, it produces result something as follows −

```
Address of var[0] = 0xbfa088b0
Value of var[0] = 10
Address of var[1] = 0xbfa088b4
Value of var[1] = 100
```

Address of var[2] = 0xbfa088b8
Value of var[2] = 200

4. Pointer to Pointer

A pointer to a pointer is a form of multiple indirection or a chain of pointers. Normally, a pointer contains the address of a variable. When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value as shown below.



A variable that is a pointer to a pointer must be declared as such. This is done by placing an additional asterisk in front of its name. For example, following is the declaration to declare a pointer to a pointer of type int −

```
int **var;
```

When a target value is indirectly pointed to by a pointer to a pointer, accessing that value requires that the asterisk operator be applied twice, as is shown below in the example −

```
#include <iostream>
 using namespace std;
 int main () {
   int  var;
   int  *ptr;
   int  **pptr;

   var = 3000;

   // take the address of var
   ptr = &var;

   // take the address of ptr using address of operator &
   pptr = &ptr;

   // take the value using pptr
   cout << "Value of var :" << var << endl;
   cout << "Value available at *ptr :" << *ptr << endl;
   cout << "Value available at **pptr :" << **pptr << endl;
   return 0;
}
```

When the above code is compiled and executed, it produces the following result −

```
Value of var :3000
Value available at *ptr :3000
Value available at **pptr :3000
```

Lab Tasks

1- What would be the output of the following function, point out the errors, if any.

```
#include <iostream>
using namespace std;

int main ()
{
  int firstvalue = 5, secondvalue = 15;
  int * p1, * p2;

p1 = &firstvalue;  // p1 = address of firstvalue
p2 = &secondvalue; // p2 = address of secondvalue
*p1 = 10;          // value pointed to by p1 = 10
*p2 = *p1;         // value pointed to by p2 = value pointed by p1
p1 = p2;           // p1 = p2 (value of pointer is copied)
*p1 = 20;          // value pointed by p1 = 20

  cout << "firstvalue is " << firstvalue << '\n';
  cout << "secondvalue is " << secondvalue << '\n';
  return 0;
}
```

2- Write a C++ program by using pointer that receives array of 5 integers from keyboard and calculate the sum, average and standard deviation of these numbers.

3- Modify the solution of task 2 in order to print the elements of the array in reverse order by using a pointer.

4- Write a program that asks the user to enter integers as inputs to be stored in the variables 'h' and 'j' respectively. There are also two integer pointers named ptrH and ptrJ. Assign the values of 'h' and 'j' to ptrH and ptrJ respectively, and display them.

5- Write a C++ program to find the max of an integral data set. The program will ask the user to input the number of data values in the set and each value. The program prints on screen a pointer that points to the max value.

6- Take input in variable and display same value by pointer.

7- Given the string "A string." Print on one line the letter on the index 0, the pointer position and the letter t. Update the pointer to pointer +2. Then, in another line print the pointer and the letters r and g of the string (using the pointer).